A Guide to GARA

May 2000[*]

[*] Please check http://www.mcs.anl.gov/qos/ for the latest version of this document.

Contents

Note: Before you read about programming with GARA, you should have at least a passing familiarity with Globus. You can learn more about Globus at http://www.globus.org. This guide gives an overview of the GARA architecture. If you need information on programming GARA, please see the Programmers Guide to GARA. *If you need more information about installing and using GARA, please see the* Administrators Guide to GARA. *If you would like more information about the research being done with GARA, please see the papers available at the Globus web site.*

Introduction

The GARA architecture provides programmers with convenient access to end-to-end quality of service (QoS) for programs. To do so, it provides mechanisms for making QoS reservations for different types of resources, including computers, networks, and disks. A reservation is a promise from GARA that an application will receive a certain level of service from a resource. For example, a reservation may promise a certain bandwidth on a network or a certain percentage of a CPU.

The GARA architecture is defined as a layered architecture with three levels of APIs and one level of low-level mechanisms:

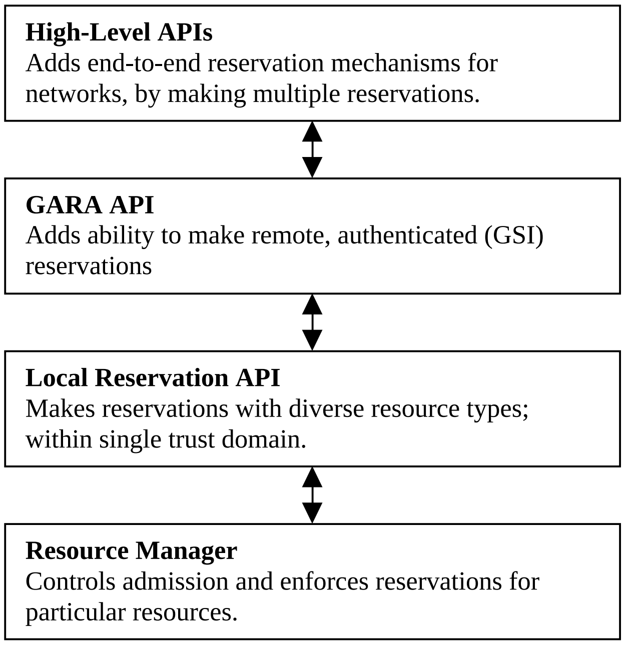

High-Level APIs
Adds end-to-end reservation mechanisms for networks, by making multiple reservations.
GARA API
Adds ability to make remote, authenticated (GSI) reservations
Local Reservation API
Makes reservations with diverse resource types; within single trust domain.
Resource Manager
Controls admission and enforces reservations for particular resources.

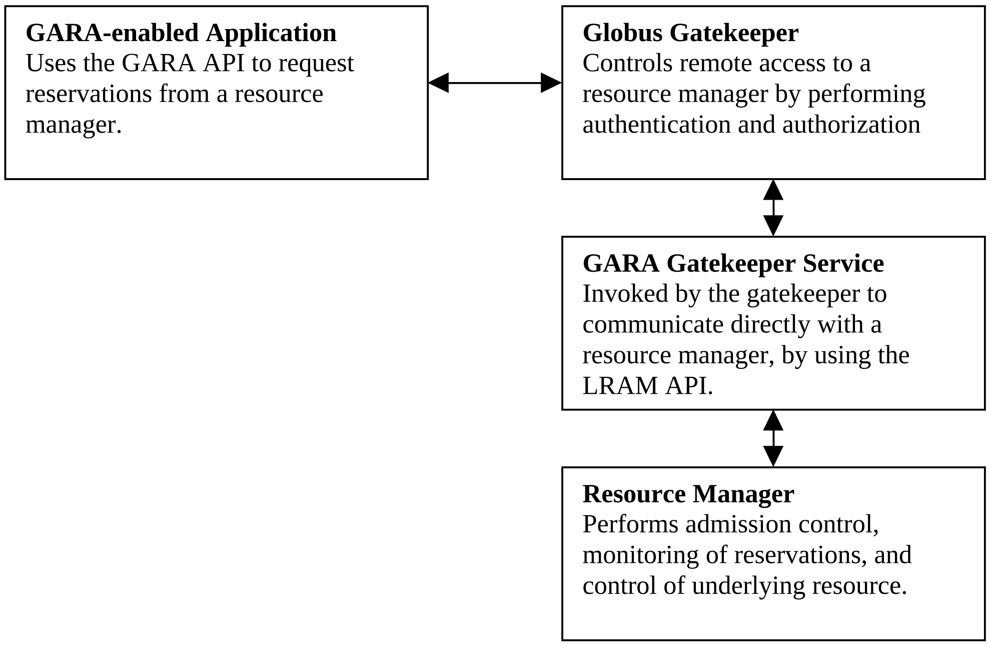
Figure 1 – GARAÕs layered architecture

Note that ÒGARAÓ refers to two things: the ÒGARA Architecture,Ó which refers to the entire diagram above, and the ÒGARA API,Ó which is the API for working with a single reservation.

The GARA API has two interesting advantages. First, it allows you to make reservations either in advance of when you need them or right at the time that you need themÑin *immediate* *reservation*. Second, you use the same API to make and manipulate a reservation regardless of the type of the underlying resource, thereby simplifying your programming when you need to work with multiple kinds of resources.

The GARA API can be considered a remote procedure call mechanism to communication with a resource manager. A resource manager controls reservations for a resource: it performs admission control and controls the resource to enforce the reservations. Some resources already have the ability to work with advanced reservations, so the resource manager is a simple program. Most resources cannot deal with advanced reservations, so the resource manager tracks the reservations and does admission control for new reservation requests. Much of the research in GARA has focused on building useful resource managers.

It is important to understand how this remote procedure call works:


GARA-enabled Application
Uses the GARA API to request reservations from a resource manager.
Globus Gatekeeper
Controls remote access to a resource manager by performing authentication and authorization
GARA Gatekeeper Service
Invoked by the gatekeeper to communicate directly with a resource manager, by using the LRAM API.
Resource Manager
Performs admission control, monitoring of reservations, and control of underlying resource.

As this figure demonstrates, when a program uses the GARA API to communicate with a resource manager, the communication does not happen directly, but happens through the assistance of the Globus gatekeeper. The gatekeeper performs three important service: authentication, to verify the identity of the person making the reservation, authorization, to verify that the person is allowed to make a reservation, and finally, the launches the gatekeeper service to handle the communication with the resource manager.

This figure demonstrates an important aspect of GARA: in order to make and use a reservation, you will need to have a Globus gatekeeper installed with a gatekeeper service properly configured, you will need to have a resource manager running, and you will need to have any underlying resource management tools installed. For example, if you wish to make reservations for CPU reservations, youŒl need the gatekeeper, the DSRT resource manager, and the DSRT scheduler process.

We will discuss each of these pieces of the GARA architecture in turn.

Resource Managers

Currently, GARA provides three different resource managers:

- A differentiated services network resource manager to provide quality of service over a network.
- A CPU resource manager that uses the Dynamic Soft Real-Time (DSRT) scheduler[1] for controlling scheduling for a processes.
- A DPSS resource manager that allows exclusive access to a DPSS server.

In the near future, new resource managers will be created to work with other resource types.

A resource manager has four important jobs:

- Admission Control: A resource manager decides whether or not each reservation can be accepted.
- Resource Configuration: A resource manager configures the underlying resource (such as routers on a network) to ensure that each reservation actually receives the quality of service that it request.
- Monitoring: A resource manager observes the underlying resource while a reservation is active. This monitoring can provide feedback to the user of the application, such as warnings that the reservation is insufficient or too large.
- Reporting: Resource managers (optionally) report the current state of the resource manager into the Metacomputing Directory Service, an LDAP directory. The information that is reported includes the total amount of the resource that can be reserved (e.g. 50 Mbps) and a list of the reservations that have been already made.

Differentiated Service Resource Manager

The differentiated services resource manager provides reservations for network bandwidth using differentiated services. Differentiated Services is well described elsewhere[2], but we will provide a brief review here.

Differentiated Services, or diffserv, is a relatively simple method for providing different types of service to packets in a network. In networks without diffserv, all packets are treated equally poorly: although the network makes its *best-effort* to deliver the packets, there is no guarantee that packets will arrive at their destination in a timely fashion, or even a guarantee that they will arrive at all. While such best-effort networks have served us well for a long time, some programs prefer better guarantees than that.

[1] DSRT was produced independently by the MONET group at the University of Illinois at Urbana-Champagne. Although DSRT has been included in the GARA distribution, you can obtain the latest version and information about DSRT from: http://cairo.cs.uiuc.edu.

[2] A good starting point for learning about differentiated services is at the home page for the IETF Differentiated Services Working Group: http://www.ietf.org/html.charters/diffserv-charter.html.

Diffserv provides guarantees by marking each packet with a *per-hop behavior (PHB)*, which indicates how the packet is to be treated. There are very few PHBs that have been created: it is not the case that each reservation has a unique marking, but that reservations are collected into aggregates, and each aggregate is treated as a whole. For example, one PHB which is used by GARA is the expedited forwarding (EF) PHB. In a router, all packets that are marked as EF as forwarded on the network before any other kinds of packets, up to some limit. If there is careful end-to-end control, EF can provide programs with assurances that their packets will get through the network quickly and with low delay.

GARA uses diffserv with EF to provide network reservations. The total amount of EF traffic is limited, so that GARA can be sure that the applications with reservations actually receive the bandwidth that they have reserved.

Currently, GARA works with Cisco routers, which are configured with scripts that use telnet. Although GARA currently relies on Cisco routers, it is very easy to replace just the small configuration scripts in order to use GARA with other underlying router types.

DSRT Resource Manager

The Dynamic Soft Real-Time (DSRT) CPU scheduler provides real-time scheduling for CPU processes. In itÕs general form, an application can reserve *x* milliseconds of time for every *y* millisecond time chunk. For example, an application may request a block of 70ms out of every 100ms. While DSRT is highly configurable and lets users select both *x* and *y*, GARA only allows users to select a percentage, such as 70% of the CPU. GARA selects y to be 1000ms and selects *x* appropriately from the percentage.

The complete DSRT distribution has been included with GARA, and more documentation is available there.

DPSS Resource Manager

GARA can provide reservations for the Distributed Parallel Storage Server. Currently, DPSS support is limited to providing exclusive access to a DPSS server. Support for DPSS is in an early stage.

Future Resource Managers

In the near future, we expect that GARA will provide resource managers for other types of underlying resources. In particular, support for the PBS and/or Maui job schedulers will likely be added by Fall 2000, and support for the GRIO disk reservation system by the end of 2000. There are also tentative plans to allow support for reserving graphic pipelines.

The Gatekeeper and Communication with Resource Managers

The Gatekeeper & Gatekeeper Service

The gatekeeper is a standard Globus component that provides authentication and authorization. All requests that come into the gatekeeper are authenticated and authorized, then forwarded onto a gatekeeper service, which is a program that is launched every time a request comes in. The gatekeeper service forwards the request to the appropriate resource manager and quits when the request has been fully handled.

At first glance, this may seem like an inefficient model since authentication and authorization could simply be built into the resource manager. However, this model has three advantages:

1. The gatekeeper can provide authentication and authorization for different services. Currently, in addition to the gatekeeper service, it also serves the Globus Resource Allocation Manager (GRAM).
2. Separating the gatekeeper service from the gatekeeper simplifies the gatekeeper. While the gatekeeper runs as root, it runs the gatekeeper service as the user who is making the request. Thus, separating out the service reduces the amount of code that runs as root.
3. The resource manager can be focused on reservations and interfacing with the resource, and does not need to deal with authentication.

The gatekeeper service use the Local Resource Allocation Manager (LRAM) API to communicate with the resource manager.

The GARA API

The GARA API provides a simple, uniform interface for making advance reservations for any type of supported resource. The API provides functions for creating, modifying, binding (claiming) canceling, and monitoring reservations.

Although the API is as uniform as possible, different resources require different parameters. For instance, a network reservation requires a bandwidth, while a CPU reservation with DSRT requires the percent of the CPU that is desired. To accommodate these different needs within a single API, the create function call accepts an Resource Specification Language (RSL) string to specify the parameters for a single reservation. For example, a network reservation might like look:

```
&(reservation-type=network) (start-time=959096044) (duration=3600)
(endpoint-a=128.135.11.40) (endpoint-b=128.135.11.90)
(bandwidth=1000) (protocol=tcp)
```

When a reservation is made, GARA provides the calling program with a reservation handle. This is a unique string that can be used to refer to the reservation in future API calls. While programs should consider the reservation handle to be opaque, it is merely a string that can be saved to disk, printed out, and passed to other program. Therefore, a reservation may be made well in advance of the time that it is used by a program that will not be using the reservation, then provided to the program that will actually use the reservation.

Because a reservation can be made in advance, it is not possible to always know all of the information that is needed to make the reservation active. For instance, a CPU reservation requires a process ID, while a network reservation requires port numbers, which are often not known in advance. Therefore, the GARA API requires that programs bind (or claim) their reservation before using by providing this information.

One reservations have been made, they can be modified. It is possible that a reservation cannot be modified, because of insufficient availability, but if a modification attempt fails, the original reservation remains in place.

Reservations can also be monitored through two mechanisms: polling and callbacks. Programs may request the status of a function at any time with a function call. In some situations, a more efficient mechanism is for the application to provide callback function. Whenever the state of a reservation changes (begins, ends, etcÉ the callback function will be automatically called.

Monitoring can provide two types of information. The first type is information about the timing of the reservation: when it begins and ends. The second type is information about the quality of the reservation. For instance, network reservations may be informed when it is noticed that the reservation is too small to meet the programÕ actual bandwidth.

For more information about the GARA API, please see the *Programmers Guide to GARA*, which can be fond at the GARA web site: http://www.mcs.anl.gov/qos/.

High-Level APIs

Because GARA provides a uniform API for working with reservations for different types of resources, it is relatively straightforward to build higher level APIs with more functionality than the vanilla GARA API. These higher level APIs can make it significantly easier to build applications.

The End-to-End Network Reservation API

The End-to-End Network Reservation (EENR) API assists users in coordinating network reservations that span multiple domains. To make an end-to-end network reservation, one reservation must be made in each domain that the network traffic will go through. When dealing with small experimental testbeds, this is generally not a difficulty. However, when working with multiple domains, it is difficult and error-prone for a programmer to decide where the reservations need to be made. Therefore, the EENR API exists to simplify this task for programmers.

Currently, the EENR API is at an early stage of development, although it does work. Future development of the EENR API may be extensive, or it may be a thin shim for interacting with a bandwidth broker such as the one being specified by the Internet 2 QBone Bandwidth Broker Work Group.[3]

Other High-Level APIs

We are currently developing a high-level API to assist users that wish to make numerous simultaneous reservations, a task we refer to as co-reservation. A co-reservation agent has three main tasks:

1. Discover which resources can be used to satisfy the users request. Usable resources may be defined in part by which resources will work (e.g. the user needs a Solaris 2.7 machine) and by when reservations are available (e.g. is the machine available between 2 and 5pm on Thursday?)
2. Make the reservations. This step may require a certain amount of backtracking if the resources are no longer available.
3. Monitor the reservations and inform the user when the reservations are available. In addition, if reservations become unavailable (because of a resource outage or because a reservation is preempted) the co-reservation agent needs to go back to steps 1 and 2.

There is not currently a co-reservation agent available for GARA, but it is under active development.

Reading more about GARA

Documentation

More documentation about GARA is available in two companion documents: the *Programmers Guide to GARA* and the *Administrators Guide to GARA*. Both of these documents can be found at the GARA web site: http://www.mcs.anl.gov/gara/.

Technical Papers

Several technical papers have been written about the design of GARA and experiments that have been done with GARA. These papers include:

[3] http://qbone.ctit.utwente.nl/BBroker/

A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation, by Ian Foster, Alain Roy, and Volker Sander. Published in the Proceedings of the Eighth International Workshop on Quality of Service (IWQoS 2000), June 2000.

A Differentiated Services Implementation for High-Performance TCP Flows, by Volker Sander, Alain Roy, and Linda Winkler. Published in the Proceedings of the TERENA Networking Conference (TNC 2000), May, 2000.

These and other papers can be found at the GARA web site: http://www.mcs.anl.gov/gara/.

Contact Information

For more information about GARA, you can refer to the GARA web page: `http://www.mcs.an.gov/qos/`.

For technical information about GARA, as well as information about collaborations and plans for future work, you should contact:

Ian Foster(foster@mcs.anl.gov)
Alain Roy (roy@mcs.anl.gov)
Volker Sander(sander@mcs.anl.gov)